REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-04-

0302

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
	Final Report	Dec 1, 99 - Nov 30, 03

4. TITLE AND SUBTITLE
Telemetry Frame Generaton

5a. CONTRACT NUMBER

5b. GRANT NUMBER
F49620-00-1-0001

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. Matthew Hudelson
Dr. William Webb

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Pure and Applied Mathematics
Washington State University

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Air Force Office of Scientific Research
4015 Wilson Blvd.
Arlington, VA 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement: Approved for public release; distribution unlimited

20040617 072

13. SUPPLEMENTARY NOTES
DODAAD CODE:
Program Manager: Dr. Neal Glassman

14. ABSTRACT

When testing an aircraft, or other vehicle, it is normal to monitor the test on the ground through the use of telemetry. In its raw for, the telemetered data can he thought of as a stream of bits. The mapping of these bits is usually done by defining repeating and periodic telemetry frames. The design of telemetry frames is mathematically difficult (more formally, this problem is known to be NP-Hard). As shown in [1], this difficulty leads to inefficient frames being designed. That is, there are many bits being sent that carry no information. Further, there is increasing demand on the spectrum and there are many ongoing efforts to use this spectrum more efficiently. The algorithm developed during this research project and described in detail in this report will generally produce frames in which more than 80% (and often more than 90%) of the bits transmitted contain information.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Matthew Hudelson
					19b. TELEPHONE NUMBER *(Include area code)*

Telemetry Frame Generation

Matthew Hudelson and William Webb
Department of Pure and Applied Mathematics
Washington State University

Research supported by AFOSR Grant #F49620-00-1-0001

Introduction

When testing an aircraft, or other vehicle, it is normal to monitor the test on the ground through the use of telemetry. In its raw for, the telemetered data can be thought of as a stream of bits. The mapping of these bits is usually done by defining repeating and periodic telemetry frames. The design of telemetry frames is mathematically difficult (more formally, this problem is known to be NP-Hard). As shown in [1], this difficulty leads to inefficient frames being designed. That is, there are many bits being sent that carry no information. Further, there is increasing demand on the spectrum and there are many ongoing efforts to use this spectrum more efficiently. The algorithm developed during this research project and described in detail in this report will generally produce frames in which more than 80% (and often more than 90%) of the bits transmitted contain information.

Notation and Definitions

Given a collection of parameters or measurands, each sampled periodically, we wish to design a telemetry frame that accommodates the measurands with as little wasted space as possible. Each measurand possesses two parameters: Its **minimum frequency** f in occurrences per second and its **wordlength** w in bits. If there are q measurands all having the same f and w, then we encode this information in the ordered triple (q,f,w). We will use the term **data set** to refer to a collection of measurands that must be accommodated together in a telemetry frame. A typical data set might be considered as a list of (q, f, w) triples, as shown in the example below:

q	f	w
4	12	14
2	12	15
1	12	17
2	12	18
8	25	1
3	25	12
3	25	16
43	50	12
14	200	12

Table 1. An Example of a Data Set

The first row of numbers in the example in table 1 indicates that this data set has four measurands each having frequency 12 and wordlength 14 bits.

The **checksum** of a data set is the total number of bits of information to be transmitted per second and can be computed by the formula

$$\text{checksum} = \sum q_i f_i w_i .$$

For the example in table 1, the checksum is 63,368.

Given a data set, a **telemetry frame** is an arrangement of the data set such that

1. Each measurand is transmitted periodically (including across frame boundaries) at a rate no lower than its minimum frequency f.
2. No two measurands are transmitted simultaneously.

For instance, the data set {(1,9,8), (2,5,4)} represents three measurands—measurand 1 has minimum frequency 9 and wordlength 8 bits while measurands 2 and 3 each has minimum frequency 5 and wordlength 4. We can accommodate these three measurands in the telemetry frame depicted in figure 1.

1	2	1	3	1	2	1	3	1	2
1	3	1	2	1	3	1	2	1	3

Figure 1. Three measurands in a telemetry frame.

To interpret this diagram, we transmit data by reading across rows. The width of each box is proportional to the number of bits (so each "1" is in an eight-bit box, while each other number is in a four-bit box.) Each number corresponds to a particular measurand. The wider boxes accommodate measurand 1, which is actually transmitted ten times per second rather then its minimum frequency of nine.

Notice that the checksum of this data set is $(1 \times 9 \times 8) + (2 \times 5 \times 4) = 112$ bits and the telemetry frame itself is 120 bits. Such computations will be used to define the **efficiency** of telemetry frames.

Standards such as [2] may place further restrictions on telemetry frames such as maximum total sizes or numbers of bits per measurand. The algorithm presented here can easily accommodate these additional restrictions, and can be adjusted to future changes in the standards.

Classes of Data Sets

We divide all input data sets into three types:

Type I – Highly Regular: A large number of measurands but very few distinct minimum frequency and word length parameter values. Furthermore, the frequencies possess few prime factors.

Type II – Ordinary: A large number of measurands with many different wordlengths and frequencies.

Type III – Pathological: Specially constructed examples that are provably hard to pack efficiently.

There exist no sharp dividing lines between the types, but the vast majority of data sets encountered in practice are of Types I and II.

Measuring Efficiency

We would like to compare the frame size found by a given algorithm to the minimum possible valid frame size, but the latter quantity is generally very difficult to compute. On the other hand, since every valid telemetry frame must contain at least the checksum number of bits, and the checksum is easy to compute, we can measure efficiency very quickly against the checksum. If we are given a data set D and generate a frame F that accommodates it, we define the **efficiency** E(D,F) by

$$E(D,F) = \frac{\text{checksum}(D)}{\text{bits in F}}$$

This number is necessarily between 0 and 1. For our {(1,9,8), (2,5,4)} example, we obtain $E(D,F) = 112/120 \approx 0.933$.

We note the following:

1. For some very simple data sets of Type III, the efficiency may be arbitrarily small, regardless of how the telemetry frame is constructed. Consider the case of a data set of the form $\{(1,f,1),(1,1,w)\}$. In this case, the minimum possible frame length (in bits) is $f(w + 1)$. This gives an efficiency of $E(D,F) \leq (f + w) / (fw + w)$ which is small whenever both f and w are large.

2. Even for Type I and Type II data sets, the efficiency of the minimum frame is almost certainly less than 1. Thus, if we can find frames with efficiencies at least 0.9 or so, we are very close to the best possible.

3. Frames may be required to contain additional structural information such as sync words and counters. These can be construed as part of the data set and are easily handled by the algorithm described herein.

Methods for Generating Frames

We distinguish two kinds of approaches.

1. Use optimization techniques to examine all possible ways to pack a frame or all ways in a large set likely to contain an optimal or near-optimal frame.
2. Use heuristics to examine only a small subset of efficient packings.

Previous work in references [3], [4], and [5] yield reasonably efficient (usually E(D,F) at least 0.75) packings for data sets of Type I. For any data set

$$D = \{(q_i, f_i, w_i): 1 \leq i \leq m\}$$

the key parameter is m, the number of distinct types of measurand. The actual number of measurands may be much higher as the q_i values may be large. As m gets sufficiently large, method A becomes impractical. One scheme commonly used to reduce m is to round all of the w_i values to standard lengths (usually multiples of 16 bits.) Another common approach is to round the f_i values to standard values (in early algorithms, these values were powers of two.) While no algorithm can avoid rounding completely, we will see that more sophisticated approaches to rounding and packing can yield surprisingly better efficiencies.

Description of Algorithm

If we are given a data set $D = \{(q_i, f_i, w_i): 1 \leq i \leq m\}$, we desire to accommodate its measurands in an efficient telemetry frame. We build our frame in a recursive fashion by using the measurands to fill "boxes". A **box** is a measurand of a given frequency and wordlength which begins empty. We fill the box using measurands still available from D. Any measurand used in a box is deleted from D. During the process we might subdivide a box into smaller boxes, attempting to fill each in succession.

To begin the process, we strategically choose a maximum box frequency *B* called the **box size** of the frame. It is typical for *B* to possess several small prime factors. Once *B* is chosen, we preprocess the measurands in D by rounding up all of the frequencies to factors of *B*. Notice that this necessitates *B* being at least as large as the highest frequency in D.

If we are given an empty box to fill, we envision being given a "demand" to fill it. A **major demand** corresponds to the requirement to fill a box with frequency *B* while a **minor demand** corresponds to having to fill a box with a frequency lower than *B*. We attempt to fulfill major demands until the data set D has been depleted.

If there are no outstanding demands and D is not yet empty, then we create a box having frequency *B* and wordlength *W* equal to the largest wordlength among measurands still in D. We now have a (*B*, *W*)–major demand that is fulfilled by the following rules:

1. If there is a (B, W) measurand available, then we use it to fulfill the demand and delete it from D.
2. If there is no (B, W) measurand available, let f be the highest frequency among measurands with wordlength W. Let F be the highest number below B such that f is a factor of F and F is a factor of B. We create B/F boxes of size (F, W), fulfilling each of these as (F, W)-minor demands. Finally, we interweave these boxes together to fill the (B, W) box.

If we envision an (F, W) box as a rectangle of height F and width W, the interweaving process can be viewed as stacking such rectangles next to one another, reading the contents horizontally, and restacking the contents in a rectangle of height B and width W in the order they are encountered. Figure 2 depicts this where $W = 1$, $F = 2$ and $B = 6$:

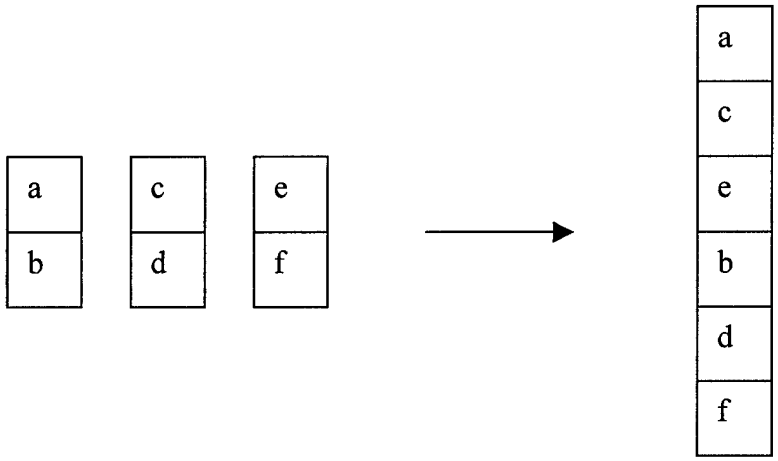


Figure 2. Interweaving three (2,1) boxes into a (6,1) box.

Fulfilling an (F, W)–minor demand is more involved. We consider the following subsets of measurands:

$S_1 = \{(f,W) : f \text{ divides } F\}$
$S_2 = \{(f,w) : f \text{ divides } F \text{ and } w < W\}$
$S_3 = \{(f,W): f \text{ does not divide } F\}$
$S_4 = \{(f,w): f \text{ does not divide } F \text{ and } w < W\}$.

The rules for fulfilling the (F,W)–minor demand are:

1. If S_1 is nonempty, choose the largest frequency among measurands in S_1. If $g = F$, then fulfill the minor demand using a corresponding measurand and delete the measurand from D. Otherwise, let G be the largest number below F such that g is a factor of G and G is a factor of F. Create F/G boxes of size (G,W) and fulfill the corresponding (G,W)–minor demands. Interweave these boxes to fulfill the original (F,W)–minor demand.

2. If S_1 is empty, but S_3 is non-empty, then round up the highest frequency in S_3 to the nearest divisor of F, and apply rule 1.
3. If S_1 and S_3 are empty, but S_2 is nonempty, then let w be the highest wordlength available from those in S_2. Create an (F,w) box and an (F,W-w) box and fulfill the corresponding demands. Concatenate the corresponding boxes as in figure 3 below.
4. If only S_4 is nonempty, then round up to the nearest divisor of F the highest frequency among those of the highest wordlength in S_4 and apply rule 3.
5. If all four sets are empty, we consider the minor demand fulfilled. This introduces empty bits into the frame.

In rule 3, we indicated the need to concatenate boxes. We show an example of this process, concatenating a (3,5) box and a (3,8) box into a (3,13) box:

Figure 3. Concatenating a (3,5) and a (3,8) box, forming a (3,13) box.

Once we have exhausted D and fulfilled all outstanding demands, we concatenate all of the boxes with frequency B as depicted in figure 4.

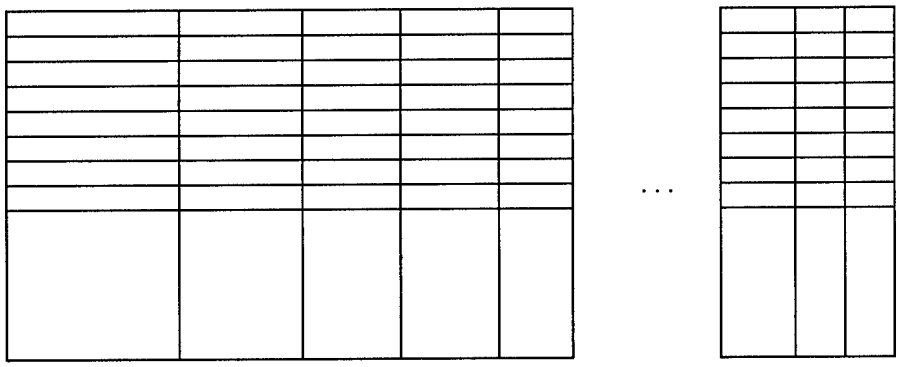


Figure 4. Concatenating B-boxes to create a frame

The frame is constructed by reading the rows horizontally. This procedure guarantees that measurands are read periodically.

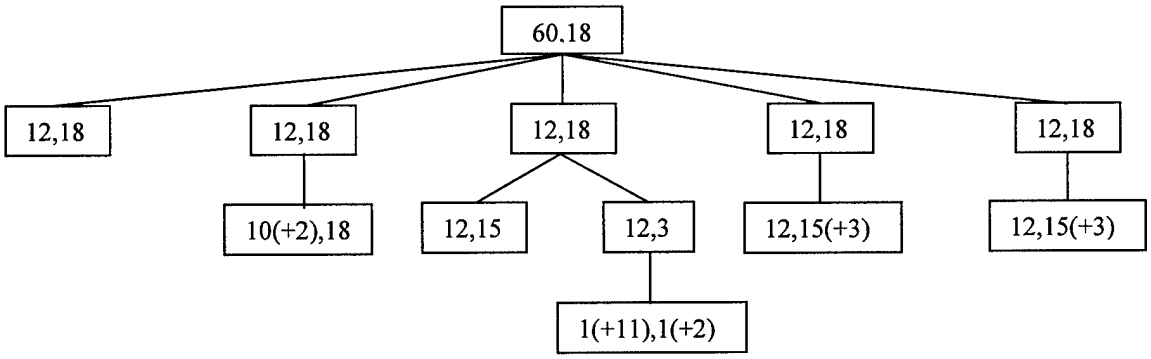
An Example in Detail

We demonstrate the algorithm, as it would be applied to the data set in Table 2, below.

q	f	w
1	12	18
1	10	18
10	12	15
4	20	15
4	12	14
2	10	14
4	20	13
32	12	13
1	10	13
6	50	12
4	25	12
23	12	12
6	10	12
1	1	1

We use a box size B = 60. We pre-process the data by rounding up all frequencies that are not factors of 60 to the next higher factor of 60.

Box 1 has an 18-bit wordlength. We initiate a (60,18) major demand. The maximum frequency among the measurands with w=18 is f=12. We will have to fulfill five (12,18) minor demands. We use the (12,18) measurand for the first of these minor demands. For the second, we use a (10,18) measurand, rounding up the 10 to 12. For the next (12,18) minor demand, we fulfill a (12,15) minor demand and a (12,3) minor demand. The first is fulfilled by a (12,15) measurand, while the second is (eventually) fulfilled by the (1,1) measurand and empty space. Finally, the last two (12,18) minor demands are each split into a (12,15) minor demand, fulfilled by a (12,15) measurand, and a (12,3) minor demand, which introduces empty space. All of this procedure is encapsulated in the following tree structure. The root of the tree is the major demand, the leaves are the measurands used to fulfill the demand. Any "+" signs indicate a frequency or word length that has been rounded up.



At this point we have fulfilled the first (60,18) major demand; the remaining measurands look like

q	f	w
7	12	15
4	20	15
4	12	14
2	10	14
4	20	13
32	12	13
1	10	13
6	60	12
4	30	12
23	12	12
6	10	12

The second box has a word length of 15 bits. We use five (12,15) measurands to satisfy the (60,15) major demand.

The third box also produces a (60,15) major demand, fulfilled by using two (12,15) measurands and three (12,14) measurands.

The fourth box is a (60,15) box, filled with three (20,15) measurands.

The fifth box is also a (60,15) box, filled with the remaining (20,15) measurand, the two (10,14) measurands, and a (20,13) measurand.

The remaining measurands are listed below.

q	f	w
1	12	14
3	20	13
32	12	13
1	10	13
6	60	12
4	30	12
23	12	12
6	10	12

The sixth box is a (60,14) box, filled with the remaining (12,14) measurand and four (12,13) measurands.

The seventh box is a (60,13) box, filled with the three remaining (20,13) measurands.

The next five boxes (8 through 12) are each (60,13) boxes, using five (12,13) measurands.

The remaining measurands are listed below.

q	f	w
3	12	13
1	10	13
6	60	12
4	30	12
23	12	12
6	10	12

Box 13 is a (60,13) box, containing the three remaining (12,13) measurands, the (10,13) measurand) and a (12,12) measurand.

Boxes 14 through 19 are (60,12) boxes each containing a single (60,12) measurand.

Boxes 20 and 21 are (60,12) boxes each containing two (30,12) measurands.

Boxes 22 through 25 are (60,12) boxes each containing five (12,12) measurands.

Box 26 is a (60,12) box containing two (12,12) measurands and three (10,12) measurands.

Box 27 is a (60,12) box containing three (10,12) measurands with unfilled room for three more (10,12) measurands.

The measurands in our data set have a checksum of 19343 bits. We have packed them into a frame requiring

$$60 \times (18 + (4 \times 15) + 14 + (7 \times 13) + (14 \times 12)) = 21060 \text{ bits.}$$

This packing has an efficiency of

$$\frac{19343}{21060} = 0.918,$$

in other words, 91.8% of the bits in the frame contain data.

Run-Time Analysis

This algorithm was implemented in C (the actual code is included in Appendix A). To analyze run-times and packing efficiencies, a collection of 2156 different data sets (provided by personnel at Edwards AFB and included in Appendix B) were processed. We summarize the outcome of this procedure in this section.

In order to compare with work done by Panten, et al., we run the algorithm after performing the following possible alterations:

1) We include a (1,32) measurand that represents the frame synch word.
2) We include a 16-bit measurand that represents a "minor-frame" counter. Its minimum frequency is given by f = checksum / 2056 rounded up to the nearest integer. The value 2056 is the maximum number of bits allowed in a minor frame and so each occurrence of this measurand delineates a minor frame.
3) We round each word length to a multiple of 16 bits (this initial step was done by Panten's group.)

Our runs were done first with (1) and (2), and subsequently with (1), (2), and (3). The graph in figure 5 shows the efficiencies produced by our algorithm. Several different efficiencies were computed, using variables C_1 = checksum without (3), C_2 = checksum after doing (3), F_1 = framesize without (3), F_2 = framesize with (3). We compared three such efficiencies in figure 5:

$$E_1 = C_1 / F_1; \quad E_2 = C_1 / F_2; \quad E_3 = C_2 / F_2.$$

The graph in figure 5 is sorted by descending values of E_1.

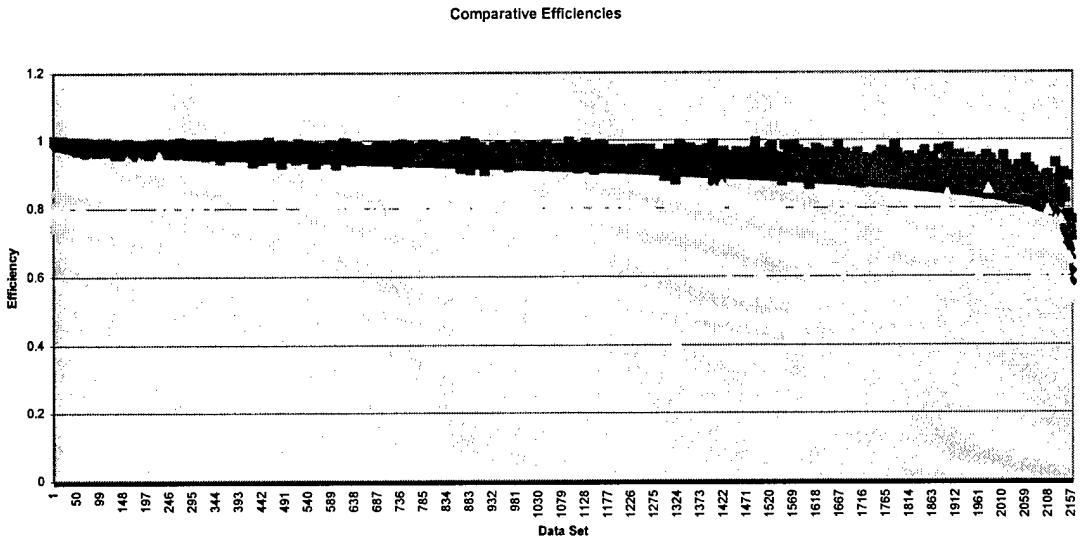


Figure 5. Efficiencies of our algorithm

In figure 5, the pink squares represent efficiencies computed using E_3, the solid blue line represents efficiencies computed using E_1, and the yellow triangles represent efficiencies computed using E_2.

In more than 60% of the runs, our algorithm produces packing efficiencies in excess of 0.90 and in over 95% of the runs, we obtain efficiencies in excess of 0.80.

While rounding wordlengths to the nearest multiple of 16 produces apparently better efficiencies (the pink squares), we note that if we divide by the revised checksum (the yellow triangles), the efficiencies are nearly always lower.

Fractional Frame Rates

Another improvement in efficiency can be realized if we drop the tacit assumption that we transmit an integer number of frames per second. The essential requirement for a measurand is that it be transmitted periodically at at least its minimum frequency. If frames are not transmitted at an integer rate, we might transmit some measurands at a non-integer frequency, still higher than the minimum frequency. We illustrate fractional frames with a greatly simplified example, using measurands (3,7,16) and (4,11,16). This example has checksum = (3)(7)(16) + (4)(11)(16) = 1040.

If the frequencies are not rounded at all, the smallest possible frame size can be shown to be (144)(16) = 2304, resulting in an efficiency below 0.50. Simple rounding of frequencies does significantly better. The obvious choices are to round 7 to 11 or to round 11 to 14.

Rounding 7 to 11 results in a minimum frame size of (77)(16) = 1232 (efficiency = 0.84). Rounding 11 to 14 results in a minimum frame size of (84)(16) = 1344 (efficiency = 0.77). Here we're assuming frames are transmitted once per second. Thus, rounding 7 to 11 is the better choice.

However, suppose we examine the data set (3,2,16) and (4,3,16) and use a box size of 6. This data set has checksum = (3)(2)(16) + (4)(3)(16) = 288 and can be accommodated perfectly in a frame having size 288. If we transmit this frame 11 times in 3 seconds, we transmit 1056 bits per second, of which 1040 of the bits contain information. This results in an efficiency of 0.98, which is a striking improvement.

Furthermore, each (2,16) measurand is transmitted 11/3 times per second, resulting in 22/3 bits per second. This is higher than the 7 bits per second minimum frequency. Also, each (3,16) measurand transmits exactly 11 bits per second at this rate, which is the minimum frequency for the second type of measurand. This example shows that unusual frequencies are not necessarily a hindrance.

We describe in more detail how to generate fractional frames. Among the original frequencies $\{f_i\}$, let $F_1 > F_2 > \ldots > F_t$ be the distinct values. We will round the F_i to new frequencies F_i^* as follows:

Let B run through the list of likely box sizes. Highly composite numbers of relatively small size are often good choices. Let $B = d_1 > d_2 > \ldots > d_s = 1$ be the divisors of B, and let $[x]_B$ denote the smallest divisor $d \geq x$. We examine fractions a/b where we will transmit a/b frames per second. The minimum frequency requirements are then

$$aF_i^*/b \geq F_i$$

for all i. We therefore set

$$F_i^* = [bF_i/a\,]_B\,.$$

Begin by setting $a/b = F_1/B$, so for example $F_1^* = B$. Each F_i^* is some divisor $d_{g(i)}$ of B. At each iteration, replace a/b by

$$\max_i \{F_i / d_{g(i)+1}\}.$$

We stop when $F_1^* = 1$. There are at most st values of a/b to examine.

We re-work our previous example, assuming a/b = 5/2. Our measurands' frequencies are multiplied by 2/5 and rounded up to the nearest factor of 20, which is our box size; the results are given in table 3 below.

q	f	w
1	5	18
1	4	18
4	10	15
10	5	15
4	5	14
2	4	14
4	10	13
32	5	13
1	4	13
6	20	12
4	10	12
23	5	12
6	4	12
1	1	1

Table 3. Adjusted frequencies for 5/2 transmission rate.

After running the algorithm on these measurands, using a box size of 20, we find these can be accommodated using the following boxes:

q	f	w
1	20	18
4	20	15
2	20	14
10	20	13
15	20	12

This frame occupies 8320 bits. Since the frame is transmitted five times every two seconds, the resulting frame transmits 20800 bits per second. This yields an efficiency of

$$\frac{19343}{20800} = 0.93$$

which is an improvement over the 0.918 efficiency computed before using the 1 frame per second transmission rate.

Appendix A. The code (in C) used to implement the algorithm.

```
#include <stdio.h>

#define MAX 80
#define MIN 10
#define LIST 80

char word[MAX];
int length_of_word;

typedef struct Measurand
        {
        int quantity;
        int frequency;
        int wordlength;
        int demand;
        } measurand ;

typedef struct Frame
        {
        int name;
        int checksum;
        int number;
        int file_checksum;
        measurand parameter[LIST];
        } frame ;


frame original,packed,best;
frame dummy;
int boxsize;

main(argc, argv)
int argc;
char *argv[];
{
    FILE *file;
    int length_of_word;
    int setname;
    int index;

    int extract_number();
    void process_data();
    void create_frame();
    void initialize_original();

    if(file = fopen(argv[1], "r"))
    {
      while(fgets(word,MAX,file))
      {
        length_of_word = strlen(word)-1;
        if((word[0]=='S')&&(word[6]=='S'))
        {
          initialize_original();
          index = 0;
```

```
            original.name = extract_number(length_of_word);
          } else {
            if((word[0]=='S')&&(word[6]=='D'))
            {
              original.number = extract_number(length_of_word);
            } else {
              if(word[0]=='C')
              {
                original.file_checksum = extract_number(length_of_word);
                original.number = index;
                create_frame();
              } else {
                process_data(length_of_word,index);
                index++;
              }
            }
          }
        }
      }
}

/*****************************************************************/
/*                                                               */
/*                      END OF MAIN PROGRAM                      */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                     BEGIN INITIALIZE ORIGINAL                 */
/*                                                               */
/*****************************************************************/

void initialize_original()
{
int i;

original.name = 0;
original.number = 0;
original.checksum = 0;
for(i=0;i<LIST;i++)
{
  original.parameter[i].quantity = 0;
  original.parameter[i].frequency = 0;
  original.parameter[i].wordlength = 0;
  original.parameter[i].demand = 0;
}

}

/*****************************************************************/
/*                                                               */
/*                     END INITIALIZE ORIGINAL                   */
/*                                                               */
```

```
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN EXTRACT_NUMBER                     */
/*                                                               */
/*****************************************************************/

int extract_number(length)
int length;
{
    int i,j;

    i = 0;
    while(((word[i]<'0')||(word[i]>'9'))&&(i<=length))
      i++;
    j = 0;
    while(((word[i]>='0')&&(word[i]<='9'))&&(i<=length))
    {
      j = 10*j + char_to_digit(word[i]);
      i++;
    }
    if(i>length)
      printf("i > length\n");
    return j;
}

/*****************************************************************/
/*                                                               */
/*                      END EXTRACT_NUMBER                       */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*                      BEGIN PROCESS_DATA                       */
/*                                                               */
/*****************************************************************/

void process_data(length,ind)
int length,ind;
{
    int i,j;

    i = 0;
    while(((word[i]<'0')||(word[i]>'9'))&&(i<=length))
      i++;
    j = 0;
    while(((word[i]>='0')&&(word[i]<='9'))&&(i<=length))
    {
      j = 10*j + char_to_digit(word[i]);
      i++;
    }
    original.parameter[ind].quantity = j;
```

```
    while(((word[i]<'0')||(word[i]>'9'))&&(i<=length))
      i++;
    j = 0;
    while(((word[i]>='0')&&(word[i]<='9'))&&(i<=length))
    {
      j = 10*j + char_to_digit(word[i]);
      i++;
    }
    original.parameter[ind].frequency = j;
    while(((word[i]<'0')||(word[i]>'9'))&&(i<=length))
      i++;
    j = 0;
    while(((word[i]>='0')&&(word[i]<='9'))&&(i<=length))
    {
      j = 10*j + char_to_digit(word[i]);
      i++;
    }

    i = j;
/*  AUSTRALIAN WORD-LENGTH RULE:  ROUND WORD LENGTHS UP TO NEAREST
   MULTIPLE OF 16    */
/*    i = ((int)((j-1)/16)+1)*16; */
    original.parameter[ind].wordlength = i;
}

/*****************************************************************/
/*                                                               */
/*                      END PROCESS_DATA                         */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN CREATE_FRAME                       */
/*                                                               */
/*****************************************************************/

void create_frame()
{
    int
box_size,largest_freq,largest_wordlength,test_checksum,max_box_size;
    int current_dummy,current_packed,leastminors;

    void print_original();
    void print_packed();
    void print_dummy();
    void print_best();
    void initialize_dummy();
    void initialize_best();
    int initialize_packed();
    int sort_original();

/*  leastminors IS THE SMALLEST ALLOWABLE NUMBER OF MINOR FRAMES  */
    leastminors = (int)(original.file_checksum / 2056);
```

```
    if(leastminors < 1)
      leastminors = 1;

/*  THESE ADDITIONAL MEASURANDS ACCOUNT FOR THE MAJOR FRAME SYNCHWORD
(wordlength = 32) AS WELL
AS FO
R THE
MINOR FRAME COUNTERS (each having wordlength = 16) */


    original.parameter[original.number].frequency = 1;
    original.parameter[original.number].quantity = 1;
    original.parameter[original.number].demand = 0;
    original.parameter[original.number].wordlength = 32;
    original.parameter[original.number+1].frequency = leastminors;
    original.parameter[original.number+1].quantity = 1;
    original.parameter[original.number+1].demand = 0;
    original.parameter[original.number+1].wordlength = 16;
    original.number = original.number + 2;
    largest_freq = sort_original(0,original.number-1);
    largest_wordlength = original.parameter[0].wordlength;
/*    print_original();     */
    box_size = largest_freq;
    test_checksum = initialize_packed(box_size);
    initialize_dummy();
    current_dummy = 0;
    current_packed = 0;
    while(current_packed < packed.number)
    {
        if(dummy.parameter[current_dummy].frequency == 0)
        {
            dummy.parameter[current_dummy].quantity = 0;
            dummy.parameter[current_dummy].frequency = box_size;
            dummy.parameter[current_dummy].wordlength =
packed.parameter[current_packed].wordlength;
        }
        dummy.parameter[current_dummy].demand = 1;
        current_packed =
fulfill_major_demand(current_packed,current_dummy,box_size);
        dummy.parameter[current_dummy].quantity++;
        dummy.parameter[current_dummy].demand = 0;
        if((current_packed <
packed.number)&&(packed.parameter[current_packed].wordlength <
dummy.par
ameter[current_dummy].wordlength))
          current_dummy++;
          dummy.number = current_dummy+1;
    }
    dummy.checksum = check_dummy_checksum();
    initialize_best();
    max_box_size = 4*largest_freq;
    if(largest_freq > 200)
      max_box_size = 3*largest_freq;
    if(largest_freq > 400)
      max_box_size = 2*largest_freq;
    for(box_size = largest_freq + 1; box_size < max_box_size;
box_size++)
```

```
    {
      test_checksum = initialize_packed(box_size);
      if(test_checksum < best.checksum)
      {
        initialize_dummy();
        current_dummy = 0;
        current_packed = 0;
        while(current_packed < packed.number)
        {
            if(dummy.parameter[current_dummy].frequency == 0)
            {
                dummy.parameter[current_dummy].quantity = 0;
                dummy.parameter[current_dummy].frequency = box_size;
                dummy.parameter[current_dummy].wordlength =
packed.parameter[current_packed].wordleng
th;
            }
            dummy.parameter[current_dummy].demand = 1;
            current_packed =
fulfill_major_demand(current_packed,current_dummy,box_size);
            dummy.parameter[current_dummy].quantity++;
            dummy.parameter[current_dummy].demand = 0;
            if((current_packed <
packed.number)&&(packed.parameter[current_packed].wordlength <
dummy
.parameter[current_dummy].wordlength))
              current_dummy++;
        }
        dummy.number = current_dummy+1;
        dummy.checksum = check_dummy_checksum();
        if(dummy.checksum < best.checksum)
          initialize_best();
      }
    }
    print_best();
}


/*****************************************************************/
/*                                                               */
/*                      END CREATE_FRAME                         */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN FULFILL MAJOR DEMAND               */
/*                                                               */
/*****************************************************************/

int fulfill_major_demand(cp,cd,b)
int cp,cd,b;
{
    int fulfill_minor_demand();
```

```
    void print_dummy();

    int q, f, w, d, m, i, c;

    q = packed.parameter[cp].quantity;
    f = packed.parameter[cp].frequency;
    w = packed.parameter[cp].wordlength;
    m = q*f;
    if(m >= dummy.parameter[cd].frequency)
    {
        d = dummy.parameter[cd].frequency/f;
        packed.parameter[cp].quantity = packed.parameter[cp].quantity -
d;
    } else {
        /*  d = (dummy.parameter[cd].frequency - m)/f;  */
        d = 2;
        while((dummy.parameter[cd].frequency % d >
0)||(dummy.parameter[cd].frequency/d % f >
0))
          d++;
        c = cp;
        for(i = 0; i < d; i++)
          c =
fulfill_minor_demand(c,cd+1,dummy.parameter[cd].frequency/d,w,1);
        d = 0;
    }
    c = 0;
    while((c < packed.number)&&(packed.parameter[c].quantity == 0))
      c++;
    return c;
}

/*****************************************************************/
/*                                                               */
/*                      END FULFILL MAJOR DEMAND                 */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*                      BEGIN FULFILL MINOR DEMAND               */
/*                                                               */
/*****************************************************************/

int fulfill_minor_demand(cp,cd,ff,ww,b)
int cp,cd,ff,ww,b;
{
    int q, f, w, d, m, i, c,settype,b1,b2,b3,b4;

    void print_packed();
    void print_dummy();

    c = cp;
    while((c<packed.number)&&(packed.parameter[c].quantity <= 0))
    {
      if(packed.parameter[c].quantity < 0)
      {
```

```
          packed.parameter[c].quantity = 0;
        }
        c++;
      }
      if(c == packed.number)
      {
        return cp;
      }



      b1 = cp-1;
      b2 = cp-1;
      b3 = cp-1;
      b4 = cp-1;
    while(c < packed.number)
      {
        if((packed.parameter[c].quantity > 0) &&
(packed.parameter[c].wordlength
== ww)&&(ff %
packed.p
arameter[c].frequency == 0)&&(b1 == cp-1))
          b1 = c;
        if((packed.parameter[c].quantity > 0) &&
(packed.parameter[c].wordlength <
ww)&&(ff %
packed.pa
rameter[c].frequency == 0)&&(b2 == cp-1))
          b2 = c;
        if((packed.parameter[c].quantity > 0) &&
(packed.parameter[c].wordlength
== ww)&&(ff >
packed.p
arameter[c].frequency)&&(ff % packed.parameter[c].frequency != 0)&&(b3
== cp-1))
          b3 = c;
        if((packed.parameter[c].quantity > 0) &&
(packed.parameter[c].wordlength <
ww)&&(ff >
packed.pa
rameter[c].frequency)&&(ff % packed.parameter[c].frequency != 0)&&(b4 ==
cp-1))
          b4 = c;
        c++;
      }

      if(b1 > cp - 1)
      {
        c = b1;
        if((packed.parameter[c].frequency ==
ff)&&(packed.parameter[c].wordlength
== ww))
        {
          packed.parameter[c].quantity--;
          if(packed.parameter[c].quantity==0) {
            return c+1;
          } else {
```

```
            return c;
          }
        }
        d = 2;
        while((ff % d > 0)||((ff/d) % packed.parameter[c].frequency  > 0))
        {
          d++;
        }
        for(i = 0;i<d;i++)
          c = fulfill_minor_demand(c,cd+1,ff/d,ww,b+1);
      } else {
    if(b3 > cp - 1)
        {
          c = b3;
          f = roundup(packed.parameter[c].frequency,ff);
          if((f == ff)&&(packed.parameter[c].wordlength == ww))
          {
            packed.parameter[c].quantity--;
            if(packed.parameter[c].quantity==0) {
              return c+1;
            } else {
              return c;
            }
          }
          d = 2;
          while((ff % d > 0)||((ff/d) % f  > 0))
          {
            d++;
          }
          for(i = 0;i<d;i++)
            c = fulfill_minor_demand(c,cd+1,ff/d,ww,b+1);
        } else {
          if(b2 > cp - 1)
          {
            c = b2;
            c =
fulfill_minor_demand(c,cd+1,ff,packed.parameter[b2].wordlength,b+1);
            c = fulfill_minor_demand(c,cd+1,ff,ww -
packed.parameter[b2].wordlength,b+1);
          } else {
            if(b4 > cp - 1)
            {
              c = b4;
              c =
fulfill_minor_demand(c,cd+1,ff,packed.parameter[b4].wordlength,b+1);
              c = fulfill_minor_demand(c,cd+1,ff,ww -
packed.parameter[b4].wordlength,b+1);
            } else {
            }
          }
        }
      }
    c = cp;
      while((c<packed.number)&&(packed.parameter[c].quantity == 0))
        c++;
      return c;
}
```

```
/*****************************************************************/
/*                                                               */
/*                      END FULFILL MINOR DEMAND                 */
/*                                                               */
/*****************************************************************/




/*****************************************************************/
/*                                                               */
/*                      BEGIN INITIALIZE PACKED                  */
/*                                                               */
/*****************************************************************/

int initialize_packed(boxsize)
int boxsize;
{
int i,j;
int roundup();
packed.name = original.name;
packed.checksum = 0;
packed.number = original.number;
for(i=0;i<LIST;i++)
{
  packed.parameter[i].quantity = original.parameter[i].quantity;
  packed.parameter[i].frequency =
roundup(original.parameter[i].frequency,boxsize);
  packed.parameter[i].wordlength = original.parameter[i].wordlength;
  packed.checksum = packed.checksum + packed.parameter[i].quantity *
packed.parameter[i].frequency *
packed.parameter[i].wordlength;
  packed.parameter[i].demand = 0;
}

return packed.checksum;

}

/*****************************************************************/
/*                                                               */
/*                      END INITIALIZE PACKED                    */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN INITIALIZE DUMMY                   */
/*                                                               */
/*****************************************************************/

void initialize_dummy()
{
```

```
int i;
int roundup();

dummy.name = original.name;
dummy.number = 0;
dummy.checksum = 0;
for(i=0;i<LIST;i++)
{
  dummy.parameter[i].quantity = 0;
  dummy.parameter[i].frequency = 0;
  dummy.parameter[i].wordlength = 0;
  dummy.parameter[i].demand = 0;
}

}

/*****************************************************************/
/*                                                               */
/*                      END INITIALIZE DUMMY                     */
/*                                                               */
/*****************************************************************/




/*****************************************************************/
/*                                                               */
/*                      BEGIN CHECK DUMMY CHECKSUM               */
/*                                                               */
/*****************************************************************/

int check_dummy_checksum()
{
int i,j;
j = 0;

for(i=0;i<LIST;i++)
{
   j = j +
dummy.parameter[i].quantity*dummy.parameter[i].frequency*dummy.parameter
[i].wordlength;
}

return j;

}

/*****************************************************************/
/*                                                               */
/*                      END CHECK DUMMY CHECKSUM                 */
/*                                                               */
/*****************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*                      BEGIN INITIALIZE BEST                    */
/*                                                               */
/*****************************************************************/

void initialize_best()
{
int i;
int roundup();
best.name = dummy.name;
best.number = dummy.number;
best.checksum = dummy.checksum;
for(i=0;i<LIST;i++)
{
  best.parameter[i].quantity = dummy.parameter[i].quantity;
  best.parameter[i].frequency = dummy.parameter[i].frequency;
  best.parameter[i].wordlength = dummy.parameter[i].wordlength;
  best.parameter[i].demand = dummy.parameter[i].demand;
}

}

/*****************************************************************/
/*                                                               */
/*                      END INITIALIZE BEST                      */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN ROUNDUP                            */
/*                                                               */
/*****************************************************************/


int roundup(small,large)
int small,large;
{
    int i;

    if(small >= large)
      return small;
    if(small <= 0)
      return 0;
    i = small;
```

```
    while(large%i > 0)
      i++;
    return i;
}

/*****************************************************************/
/*                                                               */
/*                        END ROUNDUP                            */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                        BEGIN SORT_ORIGINAL                    */
/*                                                               */
/*****************************************************************/

int sort_original(start,stop)
int start,stop;
{
int g,h,i,j,k;

    if(stop-start<=0)
      return original.parameter[start].frequency;
    if(stop-start==1)
    {
      if(original.parameter[start].wordlength <
original.parameter[stop].wordlength)
      {
        k = original.parameter[start].wordlength;
        original.parameter[start].wordlength =
original.parameter[stop].wordlength;
        original.parameter[stop].wordlength = k;
        k = original.parameter[start].frequency;
        original.parameter[start].frequency =
original.parameter[stop].frequency;
        original.parameter[stop].frequency = k;
        k = original.parameter[start].quantity;
        original.parameter[start].quantity =
original.parameter[stop].quantity;
        original.parameter[stop].quantity = k;
      }
      if(original.parameter[start].wordlength ==
original.parameter[stop].wordlength)
       if(original.parameter[start].frequency <
original.parameter[stop].frequency)
       {
        k = original.parameter[start].wordlength;
        original.parameter[start].wordlength =
original.parameter[stop].wordlength;
        original.parameter[stop].wordlength = k;
        k = original.parameter[start].frequency;
        original.parameter[start].frequency =
original.parameter[stop].frequency;
        original.parameter[stop].frequency = k;
```

```
        k = original.parameter[start].quantity;
        original.parameter[start].quantity =
original.parameter[stop].quantity;
        original.parameter[stop].quantity = k;
       }
      k = original.parameter[start].frequency;
      if(k < original.parameter[stop].frequency)
        k = original.parameter[stop].frequency;
      return k;
    }
 if(stop - start > 1)
    {
      k = (int)((start + stop)/2);
      g = sort_original(start,k);
      h = sort_original(k+1,stop);
      if(h>g)
        g = h;
      for(j = start;j<=stop;j++)
      {
        dummy.parameter[j].quantity = original.parameter[j].quantity;
        dummy.parameter[j].frequency = original.parameter[j].frequency;
        dummy.parameter[j].wordlength =
original.parameter[j].wordlength;
      }
      h = start;
      i = start;
      j = k+1;
      while(h <= stop)
      {
        if(i > k)
        {
          original.parameter[h].quantity = dummy.parameter[j].quantity;
          original.parameter[h].frequency =
dummy.parameter[j].frequency;
          original.parameter[h].wordlength =
dummy.parameter[j].wordlength;
          j++;
        }
        if((i <= k)&&(j>stop))
        {
          original.parameter[h].quantity = dummy.parameter[i].quantity;
          original.parameter[h].frequency =
dummy.parameter[i].frequency;
          original.parameter[h].wordlength =
dummy.parameter[i].wordlength;
          i++;
        }
        if((i <= k)&&(j<=stop))
          if((dummy.parameter[i].wordlength <
dummy.parameter[j].wordlength)||((dummy.parameter[i].wo
rdlength ==
dummy.parameter[j].wordlength)&&(dummy.parameter[i].frequency <
dummy.parameter[j].freque
ncy)))
          {
            original.parameter[h].quantity =
dummy.parameter[j].quantity;
```

```
            original.parameter[h].frequency =
dummy.parameter[j].frequency;
            original.parameter[h].wordlength =
dummy.parameter[j].wordlength;
            j++;
          } else {
            original.parameter[h].quantity =
dummy.parameter[i].quantity;
            original.parameter[h].frequency =
dummy.parameter[i].frequency;
            original.parameter[h].wordlength =
dummy.parameter[i].wordlength;
            i++;
          }
        h++;
      }
      return g;
    }
    return 0;
}

/*********************************************************************/
/*                                                                   */
/*                          END SORT_ORIGINAL                        */
/*                                                                   */
/*********************************************************************/



/*********************************************************************/
/*                                                                   */
/*                       BEGIN PRINT_ORIGINAL                        */
/*                                                                   */
/*********************************************************************/

void print_original()
{
    int i,j;

    j = 0;
    for(i=0;i<original.number;i++)
       j = j + original.parameter[i].quantity *
original.parameter[i].frequency
*
original.parameter[
i].wordlength;
    original.checksum = j;
    printf("SAMPLE SET %d, NUMBER OF DISTINCT MEASURANDS = %d\n CHECKSUM
= %d,
CHECKSUM FROM
FILE = %
d\n",original.name, original.number, original.checksum,
original.file_checksum);
    j = 0;
    for(i=0;i<original.number;i++)
    {
```

```
      printf(" %d, %d, %d |
",original.parameter[i].quantity,original.parameter[i].frequency,origina
l
.parameter[i].wordlength);
      if(i<original.number-1)
        if(original.parameter[i].wordlength !=
original.parameter[i+1].wordlength)

printf("\n-----------------------------------------------------------\n");
      if(i==original.number-1)

printf("\n===========================================================\n");
    }
}

/*****************************************************************/
/*                                                               */
/*                      END PRINT_ORIGINAL                       */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN PRINT_PACKED                       */
/*                                                               */
/*****************************************************************/

void print_packed()
{
    int i,j;

    j = 0;
    for(i=0;i<packed.number;i++)
       j = j + packed.parameter[i].quantity *
packed.parameter[i].frequency *
packed.parameter[i].wor
dlength;
    packed.checksum = j;
    printf("SAMPLE SET %d, NUMBER OF DISTINCT MEASURANDS = %d\n CHECKSUM
= %d,
CHECKSUM FROM
FILE = %
d\n",packed.name, packed.number, packed.checksum, packed.file_checksum);
    j = 0;
    for(i=0;i<packed.number;i++)
    {
      printf(" %d, %d, %d |
",packed.parameter[i].quantity,packed.parameter[i].frequency,packed.para
m
eter[i].wordlength);
      if(i<packed.number-1)
        if(packed.parameter[i].wordlength !=
packed.parameter[i+1].wordlength)
```

```
printf("\n-------------------------------------------------------\n");
      if(i==packed.number-1)

printf("\n=======================================================\n");
    }
}

/*********************************************************************/
/*                                                                   */
/*                        END PRINT_PACKED                           */
/*                                                                   */
/*********************************************************************/









/*********************************************************************/
/*                                                                   */
/*                        BEGIN PRINT_DUMMY                          */
/*                                                                   */
/*********************************************************************/

void print_dummy()
{
    int i,j;

    j = 0;
    for(i=0;i<dummy.number;i++)
       j = j + dummy.parameter[i].quantity *
dummy.parameter[i].frequency *
dummy.parameter[i].wordle
ngth;
    dummy.checksum = j;
    printf("SAMPLE SET %d, NUMBER OF DISTINCT MEASURANDS = %d\n CHECKSUM
= %d,
CHECKSUM FROM
FILE = %
d\n",dummy.name, dummy.number, dummy.checksum, dummy.file_checksum);
    j = 0;
    for(i=0;i<dummy.number;i++)
    {
      printf(" %d, %d, %d |
",dummy.parameter[i].quantity,dummy.parameter[i].frequency,dummy.paramet
e
r[i].wordlength);
      if(i<dummy.number-1)
        if(dummy.parameter[i].wordlength !=
dummy.parameter[i+1].wordlength)

printf("\n-------------------------------------------------------\n");
      if(i==dummy.number-1)

printf("\n=======================================================\n");
```

```
    }
}

/*****************************************************************/
/*                                                               */
/*                      END PRINT_DUMMY                          */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN PRINT_BEST                         */
/*                                                               */
/*****************************************************************/

void print_best()
{
    int i,j;
    double efficiency;

    j = 0;
    for(i=0;i<best.number;i++)
       j = j + best.parameter[i].quantity * best.parameter[i].frequency
*
best.parameter[i].wordlengt
h;
    best.checksum = j;
    printf("SAMPLE SET %d, NUMBER OF DISTINCT MEASURANDS = %d\n CHECKSUM
=
%d\n",best.name,
best.numb
er, best.checksum);
    j = 0;

    for(i=0;i<best.number;i++)
    {
      printf(" %d, %d, %d |
",best.parameter[i].quantity,best.parameter[i].frequency,best.parameter[
i
].wordlength);
      if(i<best.number-1)
        if(best.parameter[i].wordlength !=
best.parameter[i+1].wordlength)

printf("\n-------------------------------------------------------\n");
      if(i==best.number-1)

printf("\n=======================================================\n");
    }
    efficiency = ((double) original.file_checksum)/((double)
best.checksum);
    printf(" %4d %8d %8d %f\n  ",best.name,original.file_checksum,
best.checksum, efficiency);

printf("\n=======================================================\n");
```

```
}

/*****************************************************************/
/*                                                               */
/*                      END PRINT_BEST                           */
/*                                                               */
/*****************************************************************/



/*****************************************************************/
/*                                                               */
/*                      BEGIN CHAR_TO_DIGIT                      */
/*                                                               */
/*****************************************************************/


int char_to_digit(c)
char c;
{
    int i;

    if(c<'0')
      i = 0;
    if(c>'9')
      i = 0;
    if((c>='0')&&(c<='9'))
      i = (int)(c) - (int)('0');
    return i;
}

/*****************************************************************/
/*                                                               */
/*                      END CHAR_TO_DIGIT                        */
/*                                                               */
/*****************************************************************/
```